



Docket: 43876-162

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of	:	Customer Number: 20277
	:	
HANSEN et al.	:	Confirmation Number: 5073
	:	
Application No.: 10/757,851	:	Group Art Unit: 2183
	:	
Filed: January 16, 2004	:	Examiner: Eric Coleman
	:	
For: METHOD AND SOFTWARE FOR PARTITIONED FLOATING-POINT MULTIPLY-ADD OPERATION		

DECLARATION OF KORBIN VAN DYKE

I, Korbin Van Dyke, state that:

Personal Background

1. I am an electrical/computer engineer with a BS and an MS in Electrical Engineering and Computer Science, and more than 20 years of industry and academic experience in computer architecture, processor microarchitecture, and logic design of complex VLSI systems, particularly microprocessors. I am a co-inventor on more than 50 issued U.S. patents, relating to a variety of subjects, including multiple ISA execution, hardware optimization for binary translation, video decoding ISA optimization, compatible ISA implementation techniques, speculative instruction execution, branch prediction, compatible segmentation and paging, standard PC sub-system virtualization, address generation, and logarithmic calculation.

2. I received a Master's of Science degree in Electrical Engineering and Computer Science (MS EECS) from the University of California (UC) Berkeley in 1982. Prior to receiving my Master's of Science degree, I received a Bachelor's of Science degree in Electrical Engineering and Computer Science from UC Berkeley in 1980.

3. While working toward my MS EECS degree, I also worked at UC Berkeley as a Research Assistant on the logic specification, logic verification, functional simulation, circuit design, and layout verification of an implementation of a Reduced Instruction Set Computer (RISC-I).

Declaration of Korbin S Van Dyke

4. After graduating from UC Berkeley, I worked at various engineering and management positions from 1982 through 2001 that included involvement in the specification, implementation, and debugging of complex computer architecture systems. Further selected details of my work during this period are set forth below in paragraphs 5 through 8.

5. From June 1982 to October 1987, I worked at VLSI Technology in San Jose, California as a VLSI Systems Engineer. I specified and designed a memory interface for a multi-mode display interface chip. I defined the instruction set architecture and microarchitecture of a programmable digital signal processor (DSP), and led a team that built and evaluated a microprocessor implementation of the DSP.

6. From October 1987 to May 1996, I worked at Nexgen Microsystems and remained with the company after it was acquired by Advanced Micro Devices (AMD) in January 1996, in various offices in San Jose and Milpitas, California. I designed the microarchitecture and logic details of the pipeline control for a superscalar, out-of-order, x86-compatible, multi-chip microprocessor, the Nx586. I also micro-architected and logic designed the branch prediction and multiple stream instruction fetch hardware of the Nx586. I oversaw various aspects of the physical design of the Nx586 (and its predecessors), including synthesis from Verilog code, place and route, timing closure, and layout versus schematic/netlist verification. I led a team that verified architectural correctness of the Nx586, and I led a team that debugged the Nx586 multi-chip silicon. I investigated several micro-architectural tradeoffs with respect to future high-performance x86 processors.

7. From May 1996 to May 2000, I worked at Chromatic Research and remained with the company after it was acquired by ATI Research Silicon Valley in November 1998, in Sunnyvale, and then Santa Clara, California. I oversaw the development and made personal contributions to the instruction set architecture of a dual-instruction-set processor (x86-compatible and RISC), including specifications for efficient transitions between the instruction sets, architectural hooks for efficient dynamic binary translation from the x86 instruction set to the RISC instruction set, and SIMD arithmetic and special hardware assists for media processing operations. I led a team that developed the pipeline control section of the processor (including Verilog coding and synthesis), and oversaw a team that developed the physical design of portions of the processor (place and route).

8. From June 2000 to December 2001, I worked as the Director of VLSI Development at XStream Logic and remained with the company as an Architect (after the company changed its name to Clearwater Networks in October 2001) in Los Gatos, California. As a VLSI Director, I led a team that was responsible for implementing a complex network processor from a set of specifications to silicon, using a simulatable high-level language description of the processor as a starting point. The team used a tool set that enabled automatic conversion of the high-level language into Verilog for simulation and synthesis, and then placement and routing. As an Architect, I defined and documented various architectural approaches for future network processors.

9. I am currently a sub-contractor of PatentVentures, an intellectual property consulting company with offices in San Jose, California and Austin, Texas. I have been a sub-contractor for or employed by PatentVentures as an intellectual property consultant since January

Declaration of Korbin S Van Dyke

2002 and became a registered U.S. Patent Agent (Reg. No. 52,313) in August 2002. I am also owner/operator of Van Dyke Consulting (formerly known as Korbin S Van Dyke Consulting), since May 2004, an intellectual property and technical services consulting company with an office in Sunol, California.

10. A copy of my CV is attached as Exhibit A.

Summary of My Opinions

11. In preparation of this declaration I have reviewed U.S. Patent Application Serial No. 10/757,851. I have also reviewed U.S. Patent Nos. 6,295,599 and 5,742,840 (respectively the '599 and '840 patents) that the 10/757,851 patent application indirectly claims priority to, as well as appendices to the '599 and '840 patents (the Zeus and Terpsichore System Architecture manuals, respectively, and hereinafter referred to respectively as the Zeus and the Terpsichore manuals). I have reviewed the Office Action for the 10/757,851 patent application mailed on February 21, 2007, including the paragraphs on pages 10-11 that discuss the Response to Arguments and particularly the Examiner's conclusion that the priority for the claimed invention does not extend to the '599 or the '840 patents, since features of the claimed invention are not taught or supported by the '840 or '599 patents. My understanding is that the features of the claimed invention are taught and supported by complying with the written description requirement and the enablement requirement. My understanding of the written description requirement is that a patent disclosure must describe the claimed invention in sufficient detail that one of ordinary skill in the art can reasonably conclude that the inventor had possession of the claimed invention at the time of filing the patent disclosure. My understanding of the enablement requirement is that the patent disclosure must contain sufficient information regarding the subject matter of the claims to enable one of ordinary skill in the pertinent art to make and use the claimed invention. I further understand that whether the enablement requirement is met depends on whether undue experimentation is necessary for one of ordinary skill in the art to practice the invention in light of the patent disclosure.

12. Based on my review of the materials identified in paragraph 11, it is my opinion that:

(i) the disclosure of the '599 patent and the '840 patent each would lead one of ordinary skill in the art to reasonably conclude that the inventors were in possession of the claimed (as amended) "decoding and executing instructions that instruct a computer system to perform operations", "at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands, the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands", "at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands", and

Declaration of Korbin S Van Dyke

“operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register, each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision, and a catenated result having a plurality of partitioned fields for the plurality of floating point values” elements of the 10/757,851 patent application as of the August 24, 1999 filing date of the ‘599 patent and further as of the August 16, 1995 filing date of the ‘840 patent; and

(ii) the disclosures of the ‘599 patent and the ‘840 patent each would have enabled a person of ordinary skill in the art to make and use, without undue experimentation, the claimed (as amended) “decoding and executing instructions that instruct a computer system to perform operations”, “at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands, the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands”, “at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands”, and “operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register, each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision, and a catenated result having a plurality of partitioned fields for the plurality of floating point values” elements of the 10/757,851 patent application as of the August 24, 1999 filing date of the ‘599 patent and further as of the August 16, 1995 filing date of the ‘840 patent.

13. The disclosure of the ‘840 patent provides detailed information and description that I believe would lead one of ordinary skill in the art to reasonably conclude that the inventors were in possession of the claimed (as amended) “decoding and executing instructions that instruct a computer system to perform operations”, “at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands, the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands”, “at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands”, and “operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register, each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision, and a catenated result having a plurality of partitioned fields for the plurality of floating point values” elements of the 10/757,851 patent application, and that I

Declaration of Korbin S Van Dyke

further believe would have enabled a person of ordinary skill in the art to make and use the claimed (as amended) invention without undue experimentation. On at least pages 19-21, 24-25, 29, and 136-137 of the Terpsichore manual (describing Group Floating-point Multiply and Add forms of Group Floating point Ternary instructions) there are detailed descriptions of the aforementioned claim elements.

14. The disclosure of the '599 patent provides detailed information and description that I believe would lead one of ordinary skill in the art to reasonably conclude that the inventors were in possession of the claimed (as amended) "decoding and executing instructions that instruct a computer system to perform operations", "at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands, the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands", "at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands", and "operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register, each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision, and a catenated result having a plurality of partitioned fields for the plurality of floating point values" elements of the 10/757,851 patent application, and that I further believe would have enabled a person of ordinary skill in the art to make and use the claimed (as amended) invention without undue experimentation. On at least pages 14-16, 19-20, 23-24, and 264-266 of the Zeus manual (describing Ensemble Multiply Add Floating-point forms of Ensemble Inplace Floating-point instructions) there are detailed descriptions relating to the aforementioned claim elements.

15. A detailed explanation of the basis for my opinions is set forth in the remainder of this declaration.

Detailed Basis for My Opinions

Level of Ordinary Skill in the Art:

16. Based on my review of the 10/757,851 patent application (referred to as the "media processor patent application"), I believe that the media processor patent application pertains to the technology of microprocessor design and microprocessor systems.

17. In my opinion a person of ordinary skill in the art of microprocessor design and microprocessor systems would possess a bachelor's degree in electrical engineering or computer science and have approximately five years of experience in the field of microprocessor design and microprocessor systems. Alternatively, an equivalent person could have had a master's degree in electrical engineering or computer science and have approximately two or more years of experience in the field of microprocessor design and microprocessor systems. This person would readily understand the conceptual design of a microprocessor and a microprocessor

Declaration of Korbin S Van Dyke

execution unit. The person of ordinary skill would have had access to a library of technical publications, periodicals, and textbooks.

Analysis of the disclosure of the '599 and '840 patent disclosure:

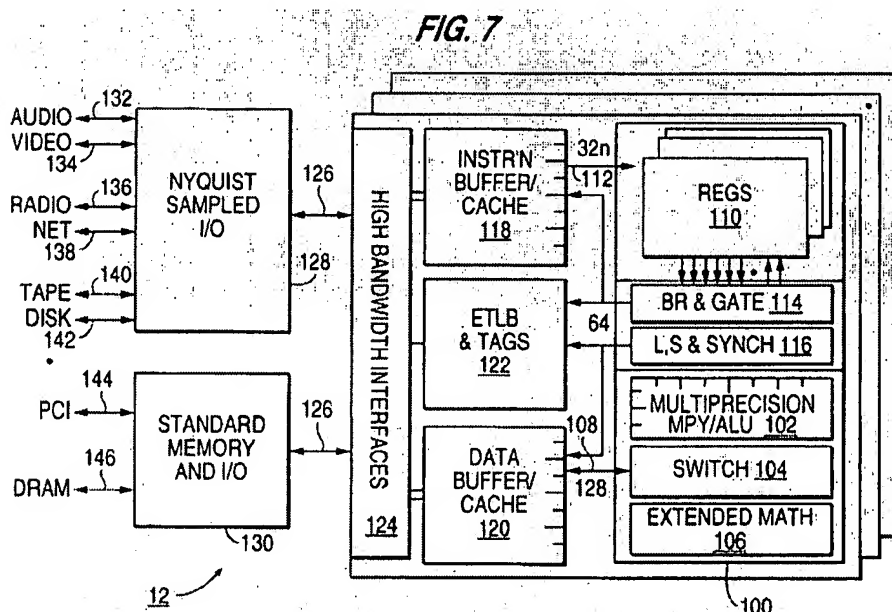
The disclosures of the '599 patent and the '840 patent each describe the claimed (as amended) "decoding and executing instructions that instruct a computer system to perform operations", "at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands, the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands", "at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands", and "operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register, each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision, and a catenated result having a plurality of partitioned fields for the plurality of floating point values" elements of the 10/757,851 patent application in sufficient detail that a person of ordinary skill in the art could reasonably conclude the inventors were in possession of the claimed invention, and that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation as of the August 24, 1999 filing date of the '599 patent and further as of the August 16, 1995 filing date of the '840 patent:

18. Claim 1 of the 10/757,851 patent application (as amended) recites various elements:
- (a) decoding and executing instructions that instruct a computer system to perform operations, at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands
 - (b) the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands
 - (c) at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands; and
 - (d) operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register

- (e) each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision
- (f) a catenated result having a plurality of partitioned fields for the plurality of floating point values

19. For brevity, the following analysis focuses on and provides details relating to the '840 patent, while reciting summary information pointing out where similar descriptive information is provided in the '599 patent.

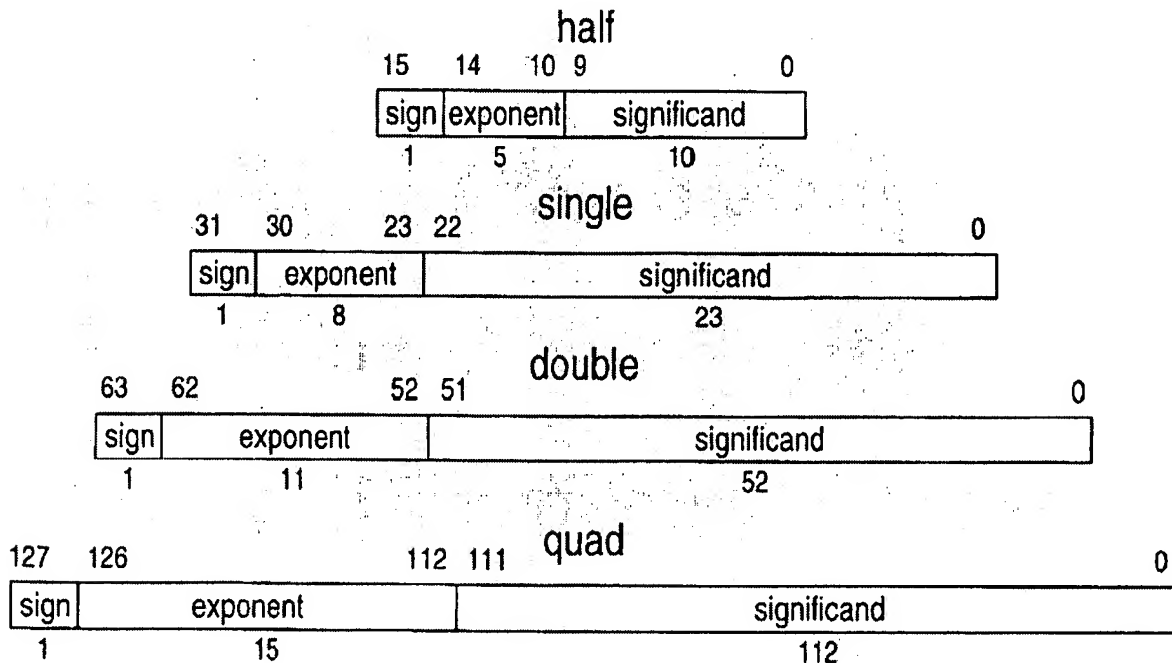
20. The '840 patent describes structure of a general purpose, programmable media processor. For example, Fig. 7 (reproduced below), illustrates an execution unit 100 having an ALU 102 that performs all logical and simple arithmetic operations (see '840, column 7, lines 8-11 and column 11, lines 50-60). Fig. 7 also illustrates a register file 110, to store and transmit data streams to and from the execution unit 100, that includes 64 general purpose registers (see '840, column 12, lines 56-64). Fig. 7 also illustrates a combined instruction buffer/cache 118 to store instructions (see '840, column 16, lines 51-67), and data buffer/cache 120 to store data received to and from the execution unit 100 and the register file 110 (see '840, column 17, lines 1-14). A unified stream of media data is processed by storage into the register file 110, and multi-precision arithmetic operations are performed on the media data. The operations include Boolean, integer, and floating-point mathematical operations (see '840, column 5, lines 47-53). Floating-point addition, subtraction, multiplication, division, and square root are supported in hardware ('840, column 15, lines 57-59). Several pipeline organizations of the general purpose media processor are described (see '840 Fig. 11 and column 17, lines 15-52, as well as '840 Fig. 12 and column 17, lines 53-67). Similarly, the '599 patent describes structure of a general purpose, programmable processor for broadband applications (see, for example, Fig. 1 and associated descriptive text, such as column 4, line 10 to column 5, line 56).



Declaration of Korbin S Van Dyke

21. The '840 patent recites that an instruction set for the general purpose media processor is described by the Microfiche Appendix (see '840 column 13, lines 21-24). The Microfiche Appendix of the '840 patent is referred to herein as the Terpsichore manual (selected pages of the Terpsichore manual are attached as Exhibit B). Similarly, the '599 patent includes and refers to a Microfiche Appendix that describes, for example, various pipeline organizations and an instruction set for a general purpose processor for broadband applications. The '599 patent Appendix is referred to herein as the Zeus manual.

22. The '840 patent further describes various floating-point data sizes such as 16, 32, 64, and 128 bits (see '840, column 15, lines 62-65, and Fig. 9b, reproduced below). The Terpsichore manual, for example on pages 19-21, describes the various floating-point data sizes as designed to satisfy ANSI/IEEE standard 754-1985. Similarly, the Zeus manual, for example on pages 14-16, provides similar information.



23. The '840 patent provides additional description relating to floating-point hardware, such as in the Terpsichore manual on page 29, "operations supported in hardware are floating-point add, subtract, multiply, divide, and square root". Further on page 29: "operations explicitly specify the precision of the operation, and round the result to the specified precision at the conclusion of each operation", and a "single instruction provides a floating-point multiply with the result fed into a floating-point add" where the "result is computed as if the multiply is performed to infinite precision, added as if in infinite precision, then rounded". Similarly, the Zeus manual, for example on pages 23-24, has a similar description.

24. The Terpsichore manual describes all of the elements of Claim 1 (as amended), on at least pages 19-21, 24-25, 29, and 136-137 (attached as Exhibit B). Paragraphs 25-30 of this declaration discuss selected portions of those pages. Paragraphs 32-38 of this declaration describe how the elements of Claim 1 are described at least by Terpsichore manual pages 19-21,

Declaration of Korbin S Van Dyke

24-25, 29, and 136-137. Similarly, the Zeus manual describes all of the elements of Claim 1, on at least pages 14-16, 19-20, 23-24, and 264-266. Note that in the Zeus manual, group floating-point instructions are termed “ensemble” floating-point instructions.

25. The Terpsichore manual describes several variations of Group Floating-point Ternary instructions, including several Group Floating-point Multiply and Add forms: “GF.MULADD.16”, “GF.MULADD.32”, and “GF.MULADD.64”, having respective ‘prec’ (or precision) values of 16, 32, and 64, as described on page 136, and reproduced below (annotations added):

Group Floating-point Ternary

These operations perform floating-point arithmetic on three groups of floating-point operands contained in registers.

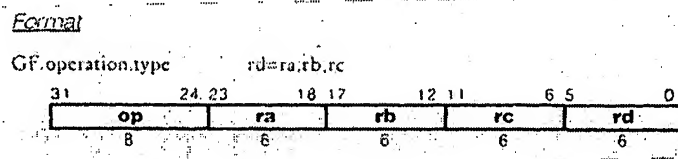
Operation codes

→ GF.MULADD.16	Group floating-point multiply and add half
→ GF.MULSUB.16	Group floating-point multiply and subtract half
→ GF.MULADD.32	Group floating-point multiply and add single
→ GF.MULSUB.32	Group floating-point multiply and subtract single
→ GF.MULADD.64	Group floating-point multiply and add double
→ GF.MULSUB.64	Group floating-point multiply and subtract double

	op	prec		
multiply and add	MULADD	16	32	64
multiply and subtract	MULSUB	16	32	64

One of ordinary skill in the art would readily understand that instances of instructions, including the foregoing Group Floating-point Multiply and Add instructions, are, under some circumstances, stored in combined instruction buffer/cache 118 of Fig. 7.

26. The Terpsichore manual, on page 136, describes an instruction format for the Group Floating-point Multiply and Add instructions, reproduced below:



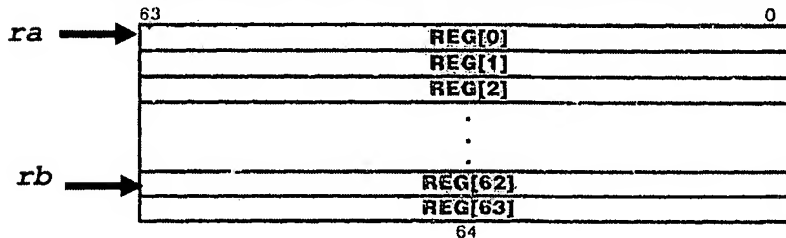
The operands of the Group Floating-point Multiply and Add instructions include ‘ra’, ‘rb’, ‘rc’, and ‘rd’. As described in more detail in paragraphs 27-29 of this declaration, contents of registers specified by the ‘ra’, ‘rb’, and ‘rc’ operands are interpreted as respective collections of partitioned floating-point operands. The partitioned floating-point operands relating to ‘ra’ and ‘rb’ are pairwise multiplied together and the multiplication results are then pairwise added to the partitioned floating-point operands relating to ‘rc’. The addition results are concatenated and then stored in a register specified by the ‘rd’ operand.

Declaration of Korbin S Van Dyke

27. The Terpsichore manual, on page 24, describes the registers referenced as operands to instructions, as reproduced below (with annotations illustrating examples of an '*ra*' operand of '0' and an '*rb*' operand of '62'):

General Registers

Terpsichore user state includes 64 general registers. All are identical; there is no dedicated zero-valued register, and there are no dedicated floating-point registers.



The foregoing registers are included in register file 110 of Fig. 7 of the '840 patent.

28. The Terpsichore manual, on pages 136-137, describes a definition of Group Floating-point Ternary instructions, including several forms of Group Floating-point Multiply and Add instructions, reproduced below (with an annotation highlighting decoding of the Group Floating-point Multiply and Add forms):

Definition

```

def GroupFloatingPointTernary(op, prec, ra, rb, rc, rd) as
  a ← RegRead(ra, 128)
  b ← RegRead(rb, 128)
  c ← RegRead(rc, 128)
  for i ← 0 to 128-prec by prec
    ai ← F(prec, a, prec-1, i)
    bi ← F(prec, b, prec-1, i)
    ci ← F(prec, c, prec-1, i)
    case op of
      GF.MULADD:
        di ← (ai * bi) + ci
      GF.MULSUB:
        di ← (ai * bi) - ci
    endcase
    di+prec-1..i ← PackF(prec, di)
  endfor
  RegWrite(rd, 128, c)
enddef

```

Declaration of Korbin S Van Dyke

29. The definition for the Group Floating-point Multiply and Add instructions, from pages 136-137 of the Terpsichore manual, would be understood by one of ordinary skill in the art as described by the following sub-paragraphs (and so would similarly the definition within pages 264-266 of the Zeus manual). The definition from pages 136-137 of the Terpsichore manual is reproduced below (with annotations):

```

Definition
def GroupFloatingPointTernary(op, prec, ra, rb, rc, rd) as
  [1] → {
    a ← RegRead(ra, 128)
    b ← RegRead(rb, 128)
    c ← RegRead(rc, 128)
    for i ← 0 to 128-prec by prec
      [3] → {
        ai ← F(prec, ai+prec-1, i)
        bi ← F(prec, bi+prec-1, i)
        ci ← F(prec, ci+prec-1, i)
        case op of
          GF.MULADD:
            [4] → di ← (ai * bi) + ci
          GF.MULSUB:
            di ← (ai * bi) - ci
        endcase
        [5] → di+prec-1,i ← PackF(prec, di)
      }
    endfor
  }
  [6] → RegWrite(rd, 128, di)
enddef
  
```

- (a) Source operands are read from pairs of registers, as specified by the '*ra*', '*rb*', and '*rc*' operands, into variables '*a*', '*b*', and '*c*', respectively (see annotation [1]), such as including reading REG[0] into the least-significant 64 bits of '*a*' and REG[1] into the most-significant 64 bits of '*a*', when '*ra*' is 0.
- (b) A '*for*' construct (see annotation [2]) specifies a number of evaluations of elements of the construct according to a '*prec*' operand that specifies a (floating-point) precision to interpret the operands. Note that the elements of the construct (see annotation [2]) are independent of each other and therefore may be evaluated entirely in parallel for improved performance ("[t]he wider the data path 108 the more unified media data can be processed in parallel by the general purpose media processor 12", '840, column 11, line 67 to column 12, line 2), entirely sequentially, or any combination thereof, as explained more fully in paragraph 29(b)v of this declaration.
 - i. For example, if the '*prec*' operand is 16, then the '*for*' construct is evaluated with eight values for variable '*i*' (0, 16, 32, 48, 64, 80, 96, and 112), respectively. For another example, if the '*prec*' operand is 64, then the construct is evaluated with two values for '*i*' (0 and 64), respectively.
 - ii. Each evaluation begins by determining a partitioned floating-point value from each of the variables '*a*', '*b*', and '*c*' (see annotation [3]) in accordance with the '*prec*' operand. For example, if the '*prec*' operand is 16, then for the evaluation where '*i*' is 0, a first partitioned floating-point value is determined from '*a*' from the least-significant 16 bits of '*a*', or '*a*_{15..0}' as identified by the expression '*a*_{*i*+prec-1..i}' in the definition. Further in the evaluation where '*i*' is 0, second and third partitioned floating-point values are determined from the least 16 bits of '*b*' and '*c*', respectively.

Declaration of Korbin S Van Dyke

Continuing with the example, for the evaluation where '*i*' is 16, partitioned floating-point values are determined from the next-most-significant 16 bits of '*a*', '*b*', and '*c*', such that bits 31 to 16 determine the floating-point values. Further continuing with the example, for the evaluation where '*i*' is 112, partitioned floating-point values are determined from the most significant 16 bits of '*a*', '*b*', and '*c*'. For another example of construct evaluations, if the '*prec*' operand is 64, then the '*for*' construct is evaluated with two values for '*i*' (0 and 64). Partitioned floating-point values are determined in the evaluation where '*i*' is 0 as the least-significant 64 bits of '*a*', '*b*', and '*c*', and in the evaluation where '*i*' is 64 as the most-significant 64 bits.

- iii. The '*for*' construct processing continues by multiplying the partitioned floating-point values from '*a*' and '*b*' by each other, and then adding the result to the partitioned floating-point value from '*c*' (see annotation [4]). The multiplying and the adding are in accordance with floating-point multiplying and adding, respectively.
 - iv. The '*for*' construct processing completes by writing the result of the multiply and add into appropriate bits of destination variable '*d*' (see annotation [5]) in accordance with the '*prec*' operand. The appropriate bits of '*d*' are identical to the bits of '*a*', '*b*', and '*c*' that the partitioned floating-point values were determined from. For example, if the '*prec*' operand is 16, then for the evaluation where '*i*' is 0, the least-significant 16 bits (i.e. bits 15 to zero) of '*d*' are written, and for the evaluation where '*i*' is 16, the next-most-significant 16 bits (i.e. bits 31 to 16) of '*d*' are written. For the evaluation where '*i*' is 112, the most-significant bits (i.e. bits 127 to 112) of '*d*' are written.
 - v. Note that each evaluation of the '*for*' construct is independent of the other evaluations, serving to operate on different unique and non-overlapping partitioned fields of the operands. Thus each evaluation may be performed in parallel with the other evaluations, sequentially with the other evaluations, or any combination thereof.
- (c) After completion of the '*for*' construct, processing completes by writing '*d*' into a pair of registers, as specified by the '*rd*' operand (see annotation [5]). Note that the definition has a typographical error, in that the '*RegWrite*' destination register is specified as '*re*', but one of ordinary would understand that '*rd*' was intended.

One of ordinary skill in the art would readily understand that the computation of the floating-point multiplies and adds would occur in ALU 102 of Fig. 7.

30. Thus the Group Floating-point Multiply and Add instructions are described in the Terpsichore manual as interpreting contents of three source registers as respective collections of partitioned floating-point operands, two of which are multiplied together and then added to the

Declaration of Korbin S Van Dyke

third. The results are concatenated together and stored in a register specified by a fourth operand.

31. At least Terpsichore manual pages 19-21, 24-25, 29, and 136-137 describe all elements of Claim 1 (as amended), as described in more detail in paragraphs 32-38 of this declaration.

32. The element decoding and executing instructions that instruct a computer system to perform operations, at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands is described by the Terpsichore manual, for example as annotated and summarized by paragraphs 25-29 of this declaration. The MULLADD forms of the Group Floating-point Ternary instructions are clearly exemplary group floating-point instructions, and the first and second registers correspond, for example, to the registers specified by operands 'ra' and 'rb'. As discussed specifically in paragraph 29(b)ii of this declaration, the specified registers are used to determine partitioned floating-point operands.

33. The element the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands is described by the Terpsichore manual, for example as annotated and summarized by paragraphs 25-29 of this declaration. The 'prec' operand defines a precision, and is specified as part of the upper eight bits of the instruction opcode as set for the in the 'Format' section of the definition. There are no restrictions stated regarding use of instructions of varying precision with respect to one another, and therefore a first MULLADD form of a Group Floating-point Ternary instruction defining a first precision may directly precede a second such instruction defining a (different) second precision. Processing the second instruction immediately after the first would be an example of dynamically varying the defined precision. As discussed specifically in paragraphs 29(b)ii and 29(b)iv of this declaration, 'prec' is used to determine a same precision for source and destination operands.

34. The element at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands is described by the Terpsichore manual, for example as annotated and summarized by paragraphs 25-29 of this declaration. The MULLADD forms of the Group Floating-point Ternary instructions are clearly exemplary group floating-point multiply-and-add instructions, and the third register corresponds, for example to the register specified by operand 'rc'. As discussed specifically in paragraph 29(b)iii of this declaration, partitioned operands from two registers (specified by 'ra' and 'rb') are multiplied together and then added to partitioned operands from a third register (specified by 'rc').

35. The element operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register is described by the Terpsichore manual, for example as annotated and summarized by paragraphs 25-29 of this declaration. The MULLADD forms of the Group Floating-point Ternary instructions are operable to perform a floating-point multiply on operands from registers specified by 'ra' and 'rb', and then to perform a floating-point add on the results of the floating-point multiply and operands from the register specified by 'rc'.

Declaration of Korbin S Van Dyke

36. The element each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision is described by the Terpsichore manual, for example as annotated and summarized by paragraphs 25-29 of this declaration. The MULLADD forms of the Group Floating-point Ternary instructions are operable to produce results of floating-point multiply and add operations with a result precision as defined by '*prec*'. As discussed specifically in paragraphs 29(b)ii and 29(b)iv of this declaration, '*prec*' is used to determine a same precision for source and destination operands.

37. The element a catenated result having a plurality of partitioned fields for the plurality of floating point values is described by the Terpsichore manual, for example as annotated and summarized by paragraphs 25-29 of this declaration. The MULLADD forms of the Group Floating-point Ternary instructions are operable to produce results into fields of bits, the fields being of a width determined by '*prec*'. As discussed specifically in paragraph 29(b)iv of this declaration, appropriate bits of '*d*' are written, corresponding to '*d*' receiving floating-point values each having a width of '*prec*'.

38. Thus every element of Claim 1 (as amended) is described at least by the Terpsichore manual on pages 19-21, 24-25, 29, and 136-137. In addition, every element of Claim 1 is also described by the '599 patent, at least by the Zeus manual on pages 14-16 (describing floating-point data formats), pages 19-20 (describing general registers), pages 23-24 (describing floating-point arithmetic hardware), and pages 264-266 (a definition for Ensemble Inplace Floating-point instructions, including MULLADD forms).

39. Based on the above, I believe that a person of ordinary skill in the art would reasonably conclude that the disclosures of the '599 patent and the '840 patent each describe the claimed (as amended) "decoding and executing instructions that instruct a computer system to perform operations", "at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands, the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands", "at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands", and "operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register, each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision, and a catenated result having a plurality of partitioned fields for the plurality of floating point values" elements, as recited in Claim 1 of the 10/757,851 patent application, in sufficient detail that the inventors had possession of the claimed invention, as of the August 24, 1999 filing date of the '599 patent and further as of the August 16, 1995 filing date of the '840 patent, and that the disclosures of the '599 patent and the '840 patent each provide sufficient detail to enable a person of ordinary skill in the art to make and use the claimed invention of the 10/757,851 patent application without undue experimentation as of the August 24, 1999 filing date of the '599 patent and further as of the August 16, 1995 filing date of the '840 patent.

Declaration of Korbin S Van Dyke

Summary and Closing:

40. The '599 patent, including the Zeus manual, provides sufficient information in sufficient detail describing the claimed invention of the 10/757,851 patent application, that one of ordinary skill in the art would reasonably conclude that the inventors had possession of the claimed invention at the time of filing the '599 patent. Further, the '599 patent, including the Zeus manual, provides sufficient information regarding the subject matter of the claimed invention of the 10/757,851 patent application to enable one of ordinary skill in the pertinent art to make and use the claimed invention without undue experimentation. In addition, the '840 patent, including the Terpsichore manual, provides sufficient information in sufficient detail describing the claimed invention of the 10/757,851 patent application, that one of ordinary skill in the art would reasonably conclude that the inventors had possession of the claimed invention at the time of filing the '840 patent. Further, the '840 patent, including the Terpsichore manual, provides sufficient information regarding the subject matter of the claimed invention of the 10/757,851 patent application to enable one of ordinary skill in the pertinent art to make and use the claimed invention without undue experimentation. Therefore, I believe that each of the '599 patent, including the Zeus manual, and the '840 patent, including the Terpsichore manual, provide adequate written description and enablement as required by 35 USC § 112 for the claimed (as amended) "decoding and executing instructions that instruct a computer system to perform operations", "at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands, the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands", "at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating-point operands", and "operable to multiply the plurality of floating-point operands in the first and second registers and add the plurality of floating-point operands in the third register, each producing a floating-point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision, and a catenated result having a plurality of partitioned fields for the plurality of floating point values" elements of Claim 1 of the 10/757,851 patent application.

41. I have had no communication with any of the inventors of the 10/757,851 patent application (Craig Hansen and John Moussouris) relating to any material in this declaration.

42. I have been hired as a consultant in connection with procedures before the United States Patent and Trademark Office (USPTO) regarding patents and patent applications assigned to Microunity Systems Engineering, Inc., including the media processor patent application. I am being compensated for my services at the rate of \$325/hour. Other than acting as a consultant in connection with procedures before the USPTO, I have no interest or connection with Microunity Systems Engineering, Inc.

43. During my evaluation of the media processor patent application, I have been impressed by the thoroughness and overall high-quality of the Zeus and Terpsichore manuals. The manuals provide clear and unambiguous descriptions of media processing systems and are thorough and well-written. The manuals provide comprehensive descriptions of instructions in complete architectural detail. The information in the manuals would have been readily

Declaration of Korbin S Van Dyke

understood and easily accessible to software engineers coding the media processing systems, and hardware engineers implementing microprocessors for use in the media processing systems, and that is exactly what architecture reference manuals should be. This is not surprising, since the '599 patent and the '840 patent each include an architecture manual that is intended to enable hardware engineers to do exactly that -- design, build, and implement a media processor that would include circuitry for "decoding and executing instructions that instruct a computer system to perform operations", "at least some of the instructions including group floating-point instructions each operating on first and second registers partitioned into a plurality of floating point operands, the floating point operands having a defined precision and the defined precision being dynamically variable, having a defined result precision which is equal to the defined precision of the operands", "at least one group floating-point instruction being a group floating-point multiply-and-add instruction, further operating on a third register partitioned into a plurality of floating point operands", and "operable to multiply the plurality of floating point operands in the first and second registers and add the plurality of floating-point operands in the third register, each producing a floating point value to provide a plurality of floating-point values, each of the floating-point values capable of being represented by the defined result precision, and a concatenated result having a plurality of partitioned fields for the plurality of floating point values;" elements as described in the Zeus and the Torpachore architecture manuals.

44. I hereby declare that all statements made herein are of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that those statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issuing therefrom.

Date: 8/21/2007

Korbin Van Dyke: K. Van Dyke
 Address: 3343 Little Valley Rd.
Sumner, CA 94586

Declaration of Korbin S Van Dyke -- Exhibit A

Korbin S. Van Dyke
3343 Little Valley Road
Sunol, CA 94586
(925) 862-0665
mail: Korbin@KorbinVanDyke.com

Personal Statement

I am an expert in Microprocessor Logic Design, VLSI Systems, and related intellectual property. Throughout more than twenty years of involvement in the specification, implementation, and debugging of complex computer architecture systems, I've enjoyed the daily challenge of learning, focusing on details, and providing working systems to end customers. In August, 2002, I became a registered US patent agent and intellectual property consultant. I now enjoy aiding clients and their counsel in understanding these technologies, and protecting their intellectual property investments in these areas.

Experience Summary

- 04/2004-present
 - Owner/Operator, Korbin S Van Dyke Consulting, office in Sunol, California

I am assisting clients in understanding technical aspects of complex intellectual property issues with emphasis on complex digital systems corresponding to my technical expertise. I have acted on behalf of plaintiffs and defendants in various intellectual property licensing and litigation projects.

Acting as a technical expert on a successful licensing project, I analyzed a collection of more than 100 computer microarchitecture patents and related prior art. I identified potentially infringing products based on publicly available information, and produced a collection of associated claim charts. In addition, I assisted the client in drafting further claims targeting other potentially infringing products.

- 01/2002-present
 - Senior Member Technical Staff and Sub-Contractor, PatentVentures, offices in California and Texas

I am authoring complex patent applications, analyzing existing intellectual property, and expanding the scope of client patent coverage. These contributions are in the areas of complex VLSI system design, leveraging my technical expertise relevant to these systems.

- 10/2001-12/2001
 - Architect, Clearwater Networks (previously XStream Logic), Los Gatos, CA

Declaration of Korbin S Van Dyke -- Exhibit A

I was the senior technical contributor in the architecture group. I was responsible for defining, documenting, and championing the architecture of future network processor products.

- 06/2000-09/2001
 - Director, VLSI Development, XStream Logic, Los Gatos, CA

I was responsible for building and leading the VLSI team developing a complex network processor. I owned the overall chip implementation and debug schedule, resource management decisions, and was also the senior technical contributor. The scope of work included front end design, verification, back end design, and debug. I built the team from ten to 35 members, and increased the skill set from front end logic design and EDA to include circuit, micro-architecture, and verification. In addition, I lead a “virtual ops” group until the operations team came on board.

- 06/1999-05/2000
 - Senior Manager, Architecture, ATI Research Silicon Valley, Santa Clara, CA

Previous positions (at ATI):

- Manager, ATI Research Silicon Valley, Santa Clara, CA

I was responsible for the top-level CPU architectural definition and partitioning, working directly with a technical writer to specify the architecture unambiguously. This requires a micro-code programmer's model negotiated between the micro-code and hardware micro-architecture implementers and logic designers. I lead the micro-code development and performance analysis teams. I also directly managed the place and route group.

I was responsible for leading a cross-functional team to design in complete X86 compatibility. This requires investigations into ambiguous or complex cases, specifying the architecture of the solution (hardware and/or software), and coordinating the complex hardware/software solutions.

I worked with contributors across the CPU to encourage and facilitate filing patent applications covering the inventions at architectural and implementation levels.

I led a small team designing the pipeline control section of a super-scalar in-order x86 compatible processor. The team was responsible for unit definition and micro-architecture, block partitioning, RTL coding, synthesis, custom macro-specification, timing analysis and improvement, cell placement, and cell routing. The team was charged with meeting area, timing, and bug count goals. I coordinated the activities of the group and negotiated interfaces with the other three units of the processor: memory, instruction fetch/conversion, and datapaths, in addition to providing specific technical guidance and solving problems.

- 05/1996-5/1999
 - Member Technical Staff, Architect, Chromatic Research, Sunnyvale, CA (merged with ATI Technology Nov 1998)

Declaration of Korbin S Van Dyke -- Exhibit A

I drove closure of the final details of the programmer's model of a complex instruction set architecture device with multiple processors. Each processor is super-scalar (in order), dual instruction set capable (industry standard X86 plus proprietary native), with hardware optimizations to support binary translation and media operations (MPEG decoding and encoding, decryption, and 3D lighting and geometry calculations).

I was a key contributor to the team refining the product definition, system and CPU level partitioning, CPU ISA, and internal CPU micro-architecture. I personally connected logic and circuit designers with media programmers via negotiation and analysis to enable a high performance well thought out and cost effective total solution.

I examined external intellectual property to identify exposures and workarounds. I contributed directly to the creation of internal intellectual property: multiple ISA execution, hardware optimization for binary translation, video decoding ISA optimization, X86 compatible ISA implementation techniques, standard PC sub-system virtualization mechanisms, and compatible segmentation and paging methods.

- 10/1987-05/1996
 - Senior Manager, AMD (formerly Nexgen), Milpitas, CA

Previous positions (at Nexgen):

- Director Processor Development
- Project manger Architecture Development
- Senior Member Technical Staff
- Member Technical Staff

I investigated possible micro-architectures for the eighth generation x86 ISA implementation, including: overall CPU performance (using an in-house performance simulation tool), expected cycle time (based on logic design of critical paths), and required process technology.

I was a key contributor toward the first Nexgen product (Nx586) – a super-scalar, out of order, fifth generation implementation of the x86 ISA. This product briefly shipped to customers.

My individual contributions included: pipeline control logic design, branch prediction and multiple stream instruction fetch micro-architecture and logic design, lab debug, timing bug identification and resolution, performance analysis and improvement, and overall design correctness.

Management responsibilities included: architectural verification, lab debug, and eventually the entire implementation of the CPU in groups from 4 to 20 people.

- 6/1982-10/1987
 - VLSI Systems Engineer, VLSI Technology, San Jose, CA

I led a small team as an active manager in the design, implementation, and prototype evaluation of a programmable digital signal processor. The project began with the definition of the

Declaration of Korbin S Van Dyke -- Exhibit A

instruction set architecture, and culminated in a working modem demonstration constructed from the first silicon.

I specified and designed the memory interface for a multi-mode display interface chip (first silicon was used for product demonstration), reverse engineered a small CMOS standard part chip, and supported a telecommunications chip set customer design project.

- 6/1981-6/1982
 - Research Assistant, UC Berkeley, Berkeley, CA

I assisted in the NMOS logic specification, logic verification, functional simulation, circuit design, and layout verification of an implementation of a Reduced Instruction Set Computer (RISC-I).

Education

6/1982 MS, EECS UC Berkeley, Berkeley, CA GPA: 4.0/4.0
6/1980 BS, EECS UC Berkeley, Berkeley, CA GPA: 3.9/4.0

Activities

I have been inducted into: Phi Beta Kapa, Tau Beta Pi , and Eta Kappa Nu

I am a member of IEEE (since 1978)

I am a member of ACM (since 1983)

Patents

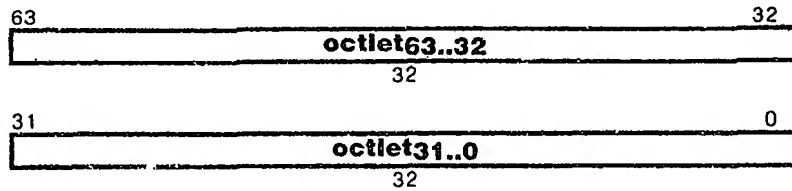
I am a co-inventor on more than 50 patents. These patents relate to a variety of subjects, including: multiple ISA execution, hardware optimization for binary translation, video decoding ISA optimization, compatible ISA implementation techniques, speculative instruction execution, branch prediction, compatible segmentation and paging, standard PC sub-system virtualization, address generation, and logarithmic calculation.

Terpsichore System Architecture

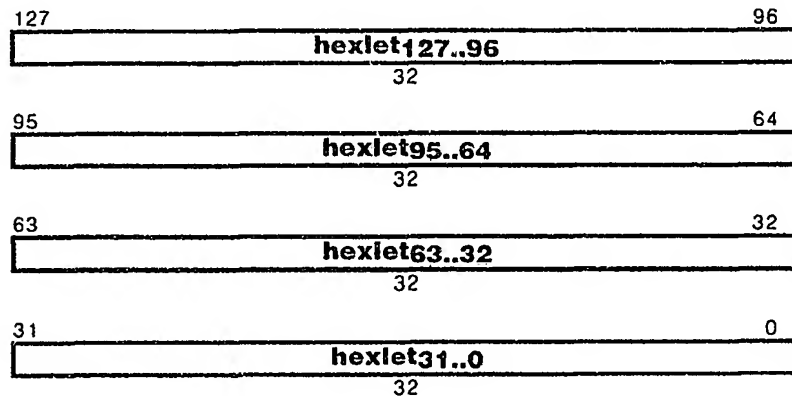
Wed, Aug 2, 1995

Octlet

An octlet is the catenation of 64 bits, and is the concatenation of eight bytes:

Hexlet

A hexlet is the catenation of 128 bits, and is the concatenation of sixteen bytes:

Address

Terpsichore addresses are octlet quantities.

Floating-point Data

Terpsichore's floating-point formats are designed to satisfy ANSI/IEEE standard 754-1985: Binary Floating-point Arithmetic. Standard 754 leaves certain aspects to the discretion of the implementor:

Terpsichore adds additional half-precision and quad-precision formats to standard 754's single-precision and double-precision formats. Terpsichore's double-precision satisfies standard 754's precision requirements for a single-extended format, and Terpsichore's quad-precision satisfies standard 754's precision requirements for a double-extended format.

Quiet NaN values are denoted by any sign bit value, an exponent field of all one bits, and a non-zero significand with the most significant bit cleared. Quiet NaN

Terpsichore System Architecture

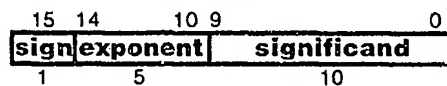
Wed, Aug 2, 1995

values generated by default exception handling of standard operations have a zero sign bit, an exponent field of all one bits, and a significand field with the most significant bit cleared, the next-most significant bit set, and all other bits cleared.

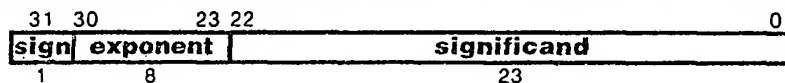
Signaling NaN values are denoted by any sign bit value, an exponent field of all one bits, and a non-zero significand with the most significant bit set.

Half-precision Floating-point

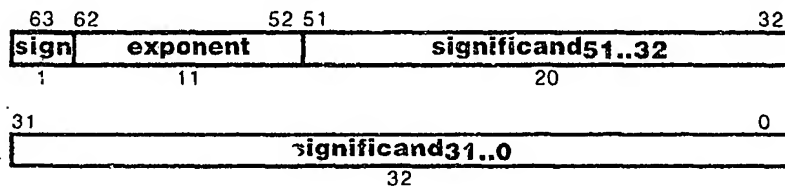
Terpsichore half precision uses a format similar to standard 754's requirements, reduced to a 16-bit overall format. The format contains sufficient precision and exponent range to hold a 12-bit signed integer.

Single-precision Floating-point

Terpsichore single precision satisfies standard 754's requirements for "single."

Double-precision Floating-point

Terpsichore double precision satisfies standard 754's requirements for "double."

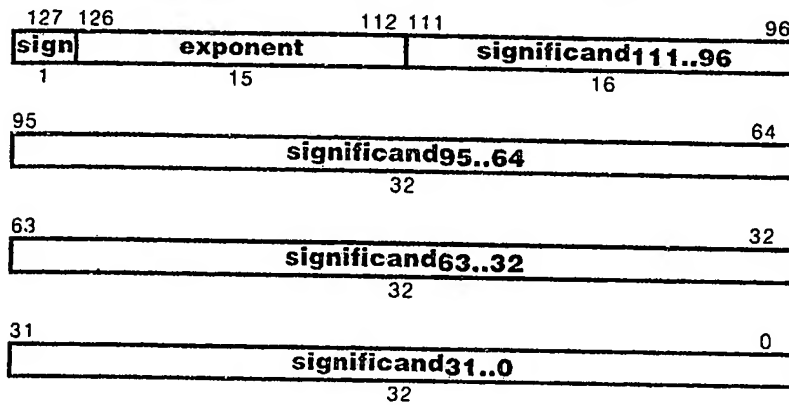


Terpsichore System Architecture

Wed, Aug 2, 1995

Quad-precision Floating-point

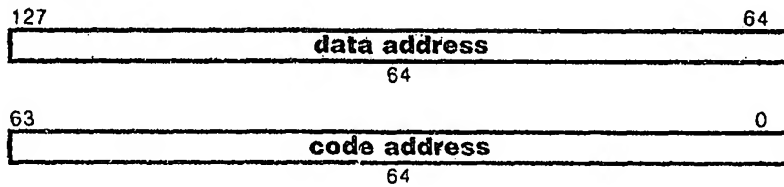
Terpsichore quad precision satisfies standard 754's requirements for "double extended," but has additional significand precision to use 128 bits.



Terpsichore System Architecture

Wed, Aug 2, 1995

The gateway contains two data items within its structure, a code address and a data address:

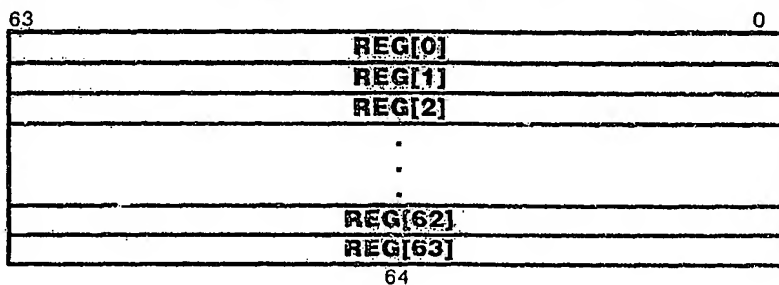


User state

The user state consists of hardware data structures that are accessible to all conventional compiled code. The Terpsichore user state is designed to be as regular as possible, and consists only of the general registers, the program counter, and virtual memory. There are no specialized registers for condition codes, operating modes, rounding modes, integer multiply/divide, or floating-point values.

General Registers

Terpsichore user state includes 64 general registers. All are identical; there is no dedicated zero-valued register, and there are no dedicated floating-point registers.



Definition

```

def val ← RegRead(rn, size)
  case size of
    64:
      val ← REG[rn]
    128:
      if rn0 then
        raise ReservedInstruction
      endif
      val ← REG[rn+1] || REG[rn]
  endcase
enddef

```

Terpsichore System Architecture

Wed, Aug 2, 1995

```

def RegWrite(rn, size, val)
  case size of
    64:
      REG[rn] ← val63..0
    128:
      if rn0 then
        raise ReservedInstruction
      endif
      REG[rn+1] ← val127..64
      REG[rn] ← val63..0
  endcase
enddef

```

Program Counter

The program counter contains the address of the currently executing instruction. This register is implicitly manipulated by branch instructions, and read by branch instructions that save a return address in a general register.

Privilege Level

The privilege level register contains the privilege level of the currently executing instruction. This register is implicitly manipulated by branch gateway and branch down instructions, and read by branch gateway instructions that save a return address in a general register.

Program Counter and Privilege Level

The program counter and privilege level may be packed into a single octet. This combined data structure is saved by the Branch Gateway instruction and restored by the Branch Down instruction.

System state

The system state consists of the facilities not normally used by conventional compiled code. These facilities provide mechanisms to execute such code in a fully virtual environment. All system state is memory mapped, so that it can be manipulated by compiled code.

Arithmetic Operations

The operations supported in hardware are floating-point add, subtract, multiply, divide, and square root. Other operations required by the ANSI/IEEE floating-point standard are provided by software libraries.

The operations explicitly specify the precision of the operation, and round the result to the specified precision at the conclusion of each operation.

A single instruction provides a floating-point multiply with the result fed into a floating-point add. The result is computed as if the multiply is performed to infinite precision, added as if in infinite precision, then rounded. This operation is a particularly good match to the needs of vector linear algebra routines.

Rounding

Rounding is specified within the instructions explicitly, to avoid maintaining explicit state for a rounding mode.

Exceptions

All the mandated floating-point exception conditions cause a trap when they occur; maintenance of sticky and other status bits may be performed using software routines. Because the floating-point inexact exception may be very frequent, this exception only occurs when specified in the instruction explicitly. Arithmetic operations may also specify that all exceptions are to be handled by default, generating special results instead of traps.

Digital Signal Processing

The Terpsichore processor provides a set of operations that maintain the fullest possible use of 64- and 128-bit data paths when operating on lower-precision fixed-point or floating-point vector values. These operations are useful for several application areas, including digital signal processing, image processing, and synthetic graphics. The basic goal of these operations is to accelerate the performance of algorithms that exhibit the following characteristics:

Low-precision arithmetic

The operands and intermediate results are fixed-point values represented in no greater than 64 bit precision. For floating-point arithmetic, operands and intermediate results are of 16, 32, or 64 bit precision.

The use of fixed-point arithmetic permits various forms of operation reordering that are not permitted in floating-point arithmetic. Specifically, commutativity and associativity, and distribution identities can be used to reorder operations. Compilers can evaluate operations to determine what intermediate precision is required to get the specified arithmetic result.



Terpsichore System Architecture

Wed, Aug 2, 1995

Group Floating-point Ternary

These operations perform floating-point arithmetic on three groups of floating-point operands contained in registers.

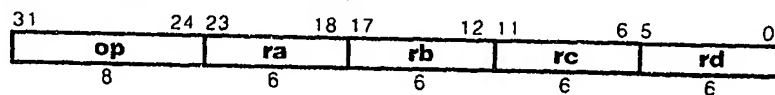
Operation codes

GF.MULADD.16	Group floating-point multiply and add half
GF.MULSUB.16	Group floating-point multiply and subtract half
GF.MULADD.32	Group floating-point multiply and add single
GF.MULSUB.32	Group floating-point multiply and subtract single
GF.MULADD.64	Group floating-point multiply and add double
GF.MULSUB.64	Group floating-point multiply and subtract double

	op	prec		
multiply and add	MULADD	16	32	64
multiply and subtract	MULSUB	16	32	64

Format

GF.operation.type rd=ra,rb,rc

Description

The contents of registers ra and rb are taken to represent a group of floating-point operands and pairwise are multiplied together and added to or subtracted from the group of floating-point operands taken from the contents of register rc. The results are concatenated and placed in register rd. The results are rounded to the nearest representable floating-point value in a single floating-point operation. Floating-point exceptions are not raised, and are handled according to the default rules of IEEE 754. These instructions cannot select a directed rounding mode or trap on inexact.

Definition

```

def GroupFloatingPointTernary(op,prec,ra,rb,rc,rd) as
  a ← RegRead(ra, 128)
  b ← RegRead(rb, 128)
  c ← RegRead(rc, 128)
  for i ← 0 to 128-prec by prec
    ai ← F(prec,ai+prec-1..i)
    bi ← F(prec,bi+prec-1..i)
    ci ← F(prec,ci+prec-1..i)
    case op of
      GF.MULADD:
        di ← (ai * bi) + ci

```

Terpsichore System Architecture

Wed, Aug 2, 1995

```
GF.MULSUB:
  di ← (ai * bi) - ci
endcase
di+prec-1.i ← PackF(prec, di)
endfor
RegWrite(re, 128, u)
enddef
```

Exceptions

Reserved instruction

Floating-point arithmetic